

Running the Quarterly Security Audit for the First Time: What Actually Broke

Ronald G. Thomas

2026-09-27



Figure 1: A row of five padlocks of different ages and finishes, one newly installed and shining among four older, weathered ones.

A control verified once is a control verified once. Nothing about that guarantees it is still true today.

Introduction

The series hub recommends a quarterly audit: a fixed checklist, run against every machine, that re-verifies controls instead of assuming they still hold. This post reports the first full run of that audit across the fleet, in the last week of September 2026, and the week of follow-up work it triggered. The audit was planned as an afternoon. It took several days, rebuilt one machine from bare metal, gave the home server a second job, and produced a list of findings in which none of the original threat model's controls failed. Full-disk encryption was on everywhere. The password store and its key were intact. The failures were in the spaces between controls.

Note

Part 8 of the security series; start with [Security Foundations for a Multi-Machine Research Fleet](#) for the overview.

The findings are told in the order they surfaced, each as a short account of what was checked, what broke, how it was found, and what changed. The lasting practice behind each one is described in depth in the post that owns the topic, and each finding links there.

Motivations

- A security posture configured once and never re-checked degrades quietly. The only way to know whether it still holds is to look.
- The fleet had grown to five machines (three Macs, a Linux laptop, and a Linux home server), and the Linux machines had never been held to the same standard as the Macs.
- Several categories of risk, notably OAuth grants and backup coverage, had no row in the original threat model, and the audit was the first time either produced a live problem.
- Backups that report success are not the same as backups that cover what they should. The audit was the first systematic comparison of the two.

Objectives

1. Run the hub's audit checklist against all five machines and record every deviation.
2. Bring the Linux laptop to parity with the Macs: disk encryption, keys, password store, dotfiles, and its own backup.
3. Compare what each backup job actually covers with what it was meant to cover, by reading logs and exit statuses rather than schedules.
4. Turn every finding into a fix, verified on every machine it touches, and a line in the fleet's written reference.



Figure 2: A single weathered rope bridge stretching between two rock faces, with a brand new, unweathered plank installed partway across.

What is a quarterly audit?

The audit is a checklist, not an investigation. The canonical copy lives in the [series hub](#); in outline it asks, for every machine: is the disk encrypted; which keys exist and where are they authorized; where do secrets live, and does anything sync or print them; which scheduled jobs exist, are they loaded, what was their last exit status, and what do their logs say; and does every repository and backup destination point where it should. The point of a fixed list is that it is run the same way each quarter, so a deviation stands out rather than being explained away.

What the first run showed is that most failures are not found by the check that targets them. They are found by a neighboring check, or by doing ordinary work carefully: reading a log line that should not be there, noticing a modified file in a repository nobody touched, or running a fix on the second machine and watching it fail.

Prerequisites

This post assumes the fleet and controls described in the earlier parts of the series:

- the five machines and the threat model ([part 1](#));
- `pass` with GPG for credentials ([part 2](#)) and on-demand loading of other secrets ([part 3](#));
- dotfiles in a git repository instead of a synced folder ([part 4](#));
- Borg, cloud copies, and Time Machine ([part 5](#));
- the home server, mercury ([part 6](#)).

No code needs to be run to follow it. The few commands shown are the checks actually used.

The Findings

1. Credentials with the wrong coverage

Checked: where every secret on every machine lives, and whether that location is backed up, synced, or both.

Broke: coverage was wrong in both directions. Two machines held their Borg passphrases only in a plain file on the machine being backed up, not in the password store. Losing either machine would have meant losing the ability to decrypt its own backup on the home server. At the same time, the files holding API keys and cloud credentials (`.Renviron`, `.env`, the AWS credentials folder) were in no backup at all, with no other copy anywhere.

Found by: walking the home folder of each machine and comparing it with each backup job's include list, item by item.

Changed: every Borg passphrase now lives in `pass`, confirmed to match the copy in use. The secret-holding files were added to every machine's Borg include list, and a backup was run and verified on each. The rules for where secrets may live are in [part 3](#).

2. The exposure, and what rotation actually involved

Checked: whether any secret had ever been somewhere it should not.

Broke: a personal settings folder on two Macs had for months been a symlink into a synced cloud folder. Everything saved there, including the GPG private key, both machines' backup passphrases, the cloud-storage OAuth tokens, and an API key, had been syncing to the provider's servers. Older leftover copies of the password store sat there too.

Found by: following the symlink. Nothing about daily use revealed it.

Changed: because access by anyone else could not be ruled out, every exposed credential was treated as compromised and replaced: a new GPG key with the store re-encrypted to it, new Borg passphrases, new SSH keys for both code hosts, new rclone OAuth tokens, a new `gh` token, and a new app password. The step that mattered most came after revoking: testing that each old credential was actually dead. Most were. A Copilot token and an old GitLab key were still live and had to be revoked separately. The migration away from the synced folder, and how to delete a shared copy safely, are in [part 4](#); rotation discipline is in [part 3](#).

3. One revocation, every machine logged out

Checked: each machine's login state for the tools that hold OAuth tokens.

Broke: during the audit two tokens were exposed by the audit itself. A routine “which account is this connected to” check on one tool printed a complete, live token to the terminal, and inspecting a plaintext token file on another machine did the same. Both were revoked at once, as a printed secret is a rotated secret. Each revocation logged out every machine using that client, not only the one being fixed, because an OAuth grant belongs to the pairing of account and client application, not to one machine's session.

Found by: the next morning's failures on machines nobody had touched.

Changed: before revoking any grant, the list of machines that will need to log in again is written down first. One machine's gh token had also been stored in a plain file rather than the Keychain; it was reissued into the Keychain like the others. [Part 3](#) covers the practice.

4. A login token inside the dotfiles repository

Checked: `git status` in the dotfiles repository on a machine where nothing had been changed.

Broke: a modified file appeared that nobody had edited. The gh tool's configuration folder is one of the folders under `~/.config` that the dotfiles setup symlinks into the repository, so every gh auth login on any machine had written that machine's live login token into a file inside the repository's working tree. It had happened on every machine. By luck it had never been committed: the only version in the history was the blank template from when the repository was created. It was a loaded gap rather than a past leak, one careless `git commit -a` away from putting a live token into the repository's history.

Found by: the unexpected modified file.

Changed: that file is now excluded from the repository on every machine, so gh keeps working with its file in the usual place while git no longer tracks it. The broader lesson is that a rule which symlinks whole configuration folders will eventually carry a credential nobody meant to share. Secret scanning on commit is the recommended guard ([part 3](#)).



Figure 3: An old woven fishing net with several large, unrepaired holes, held up against bright sky.

5. The Linux laptop, from bare metal

Checked: the Linux laptop, zeus, against the same list as the Macs.

Broke: nearly everything, in the sense that it had never been configured: no disk encryption, no shared credential access, no backup.

Changed: zeus was reinstalled with LUKS full-disk encryption (unlike FileVault, it must be chosen before partitioning), given its own keys, the GPG key copied over a trusted channel and verified to decrypt before being trusted, the password store and dotfiles cloned, and a nightly Borg backup scheduled as a systemd user timer. Two Linux differences mattered. The backup script looked for its tool in the Mac package manager's locations, which a one-line per-machine override file fixed without forking the script. And a user timer on Linux keeps running after logout only if lingering is enabled for the user; without it, the backup would have looked configured and stopped the first time the session ended. The whole process is now a written runbook rather than memory.

6. Backup coverage that vanished with a folder

Checked: what each Borg archive contains, against what it was meant to contain.

Broke: the include lists on the three Macs named the project, documents, and utilities folders indirectly, through the old synced folder they used to be symlinked into. When that folder was removed, Borg began skipping all three, logging a one-line warning each night and otherwise reporting success.

Found by: reading one morning's backup log line by line.

Changed: the lists now name the folders directly. File counts per archive went from about 24,000 to between 68,000 and 88,000 per machine, which confirmed the missing content was now captured. Borg does not fail when a listed path disappears, by design, so coverage has to be checked by reading the log ([part 5](#)).

7. Two machines racing for one cloud destination

Checked: the destinations of the nightly cloud copies.

Broke: julia and athena each mirrored their own copy of the project folder to the same cloud location. A mirror treats the other machine's version as drift to be corrected, so whichever ran last won, and neither copy was reliably complete. The same jobs had also never excluded regenerable caches; one night's upload ran fifteen hours on tens of thousands of tiny files.

Changed: every machine now has its own destination folder, and the jobs exclude the same regenerable categories Borg already skips ([part 5](#)).

8. A self-referencing link that stopped every deletion

Checked: each cloud job's log, not its schedule.

Broke: every run on both machines ended in errors. A student repository contained a link pointing to itself. The cloud copies follow links (`rsync -L`), hit the loop, and, as a safety measure, skipped every deletion for the run. Files deleted or renamed locally were never removed from any cloud copy, which kept growing, while each run looked routine.

Changed: the link was removed on every machine that had it, and the cloud copies now run cleanly. Watching for listing errors in these logs is part of the backup check ([part 5](#)).

9. Skipped runs that looked like failures, and the fix that broke joe

Checked: why one machine's cloud copy had not run for three nights.

Broke: each job takes a lock so two runs to one destination cannot overlap. When the lock was held by a long manual run, the scheduled job gave up with exit status 1 and logged nothing, indistinguishable from a real failure.

Changed: a skipped run now logs a dated line and exits with its own code:

```
/opt/homebrew/bin/flock -n -E 75 "$lock" \  
  rsync -L "$PRJ_ROOT" "$dest" "$@" || rc=$?
```

That fix then broke joe. It calls the lock tool at a fixed path, and joe had never installed it, so every cloud job there failed at once. Installing it revealed that joe had never had `rsync` configured either: its cloud jobs had been scheduled for weeks with nothing able to run. They were removed, since Borg covers joe. The fix had been tested on one machine and rolled out to three.

10. An address the router chose

Checked: a general status check of the Macs' scheduled jobs, a few days after the audit.

Broke: the home server had been restarted for routine work and the home router gave it new addresses. athena and joe still reached it by the old one, so their nightly Borg backups timed out and failed for three nights, while each scheduler recorded the failure the whole time. The first fix changed the address in the general SSH configuration and the backups still failed: the Borg script deliberately uses its own SSH configuration file, so that only the restricted backup key is offered, and the address lived there too. A day later, a check that also covered the Linux machines found that zeus had been failing the same way for four nights. The first status check had looked only at the Macs.

Changed: every Borg client now reaches the home server by its Tailscale address, which does not change, as julia already did. The new address was trusted only after confirming the server's host key matched the one recorded for the old address. The networking practice is in [part 6](#); the backup side is in [part 5](#).

The checks themselves were one line per platform:

```
# macOS: last exit status of each backup agent
launchctl list | grep -E 'borg|rclone'
# Linux: result of the last Borg run
systemctl --user show -p Result,ExecMainStatus borg-backup.service
```

11. A job that left the scheduler

Broke: julia's Borg job had dropped out of launchd, apparently at a restart, with its definition file intact, so a night passed with no run at all. A job that is not loaded produces no failure to notice.

Changed: the job was loaded again and a manual run verified. The status check now confirms that jobs are loaded, not only that their last run succeeded. A related hazard surfaced the next night: a Mac's 02:00 job can fire during a brief dark wake before the network is up and fail, so a backup script should wait or retry for connectivity before giving up ([part 5](#)).

12. Archives that never expired

Checked: how much space the Borg repositories used, and why.

Broke: nothing had ever pruned an archive. Every repository grew without limit.

Changed: a retention policy (two weeks of dailies, two months of weeklies, a year of monthlies, three years of yearlies) is now applied by hand each quarter. Automating it would require each repository's passphrase on the machine that stores it, which is exactly what the design prevents: the storage host can append but cannot decrypt. The manual step is the design working, not a gap in it ([part 5](#)).

13. A second service and no firewall

Changed situation: athena's Time Machine drive moved to the home server and was shared over Samba, giving a single-purpose machine a second job.

Broke: the server had no firewall running. Its security had rested on the narrow restricted backup keys and on nothing else listening. That was an implicit assumption, never a decision, and it stopped holding when a second service appeared.

Changed: ufw now denies incoming connections by default, trusts the Tailscale interface, and allows SSH and Samba only from the home network ([part 6](#)).

14. Time Machine: a file system that could not be checked, and a missed checkbox

Broke, twice. The external drive was first formatted HFS+, chosen deliberately because both macOS and Linux can read it. Within a week the drive dropped off USB in the middle of a write, and the Linux HFS+ checker could not verify the volume afterward. A backup volume that cannot be checked on the machine it is attached to is not one to trust, so it was reformatted ext4, a documented choice for this setup. Separately, the first Time Machine backup to the share was created without encryption, which Time Machine offers only when a destination is first added. It had reached 258 GB of one Mac's files, unencrypted, on another machine before the omission was noticed.

Changed: the unencrypted backup was deleted, the freed space trimmed, and the backup restarted with encryption on and its password stored in `pass`. The setup is described in [part 6](#) and the policy in [part 5](#).

15. A share that listened only to itself

Broke: after the server restart in finding 10, Samba was configured to listen on one named Wi-Fi adapter, started before that adapter had an address, and listened only on the server's loopback address. The share was unreachable from the network and nothing reported an error. The server also turned out to have two adapters on the same network, with Samba bound to only one, so connecting by name reached the refused address about half the time.

Found by: a direct write test on the server succeeded while athena could not connect; checking what Samba was actually listening on explained why.

Changed: Samba now waits for the network at boot and listens on both adapters. It is tested from another machine after a restart, not only on the server itself ([part 6](#)).

16. Making a full backup affordable

Checked: why athena's Time Machine configuration excluded applications and system libraries.

Changed: rather than excluding them, the installed applications on all three Macs were cut back to a research minimum. A cleanup script removed each application's support files while refusing to touch anything carrying the identifier of an application being kept, which mattered: one office

suite's uninstall list would otherwise have taken a kept word processor's data with it. The full TeX distribution was replaced with per-user TinyTeX and a package list in the dotfiles repository, and the remaining applications are recorded in a Brewfile. With a small application set, Time Machine backs up everything except data that is regenerable or held elsewhere, and a reinstall has a list to work from.

17. One login for git

Checked: how each repository authenticates to the code host.

Broke: nothing, yet, but the mix was a liability: most project repositories used SSH keys while the dotfiles repository used the `gh` tool's login.

Changed: project repositories moved to HTTPS remotes authenticated through the `gh` credential helper, whose token sits in the Keychain. A script rewrote only remotes pointing at the code host (over a hundred on the primary workstation) and confirmed along the way that no confidential, local-only repository had acquired such a remote. Two consequences were worth knowing in advance: the token needs the `workflow` scope before it can push changes to automated workflow files, and a command run over a plain SSH session into a Mac cannot reach the Keychain login, even though scheduled jobs in the logged-in session can. The same pass caught a blog post folder still pointing at the template repository it was copied from; it now has its own.



Figure 4: A single burning candle positioned so its flame lights the next candle in an otherwise unlit row.

Things to Watch Out For

- **Revoking an OAuth grant logs out every machine using that client.** List the machines that will need to log in again before revoking.
- **Some tools print raw secrets on an ordinary status check.** Know which ones, and treat anything printed as exposed.
- **Symlinking whole configuration folders into a repository sweeps in files nobody chose,** including login tokens. Check `git status` on machines where nothing changed.
- **Borg does not fail when an included path disappears.** Read the log, and check file counts after any folder move.
- **Two machines mirroring to one destination race; they do not back each other up.** Give each machine its own destination.
- **One self-referencing link can make rclone skip every deletion,** while the run still looks routine.
- **A skipped run and a failed run should not share an exit status.**
- **An address assigned by the router is not an identity.** Reach always-on machines by a stable address, and know every file that names them; a restricted key's separate SSH configuration is easy to forget.
- **A job that is not loaded produces no failure.** Check that scheduled jobs are loaded, not only that they last succeeded.
- **Options that can only be chosen at setup, such as Time Machine encryption, will eventually be missed at setup.** Verify the result immediately.
- **A service bound to a named interface can start before the interface exists.** Make it wait for the network, and test from another machine after restarts.
- **A single-purpose machine's security is often implicit.** When it gains a second purpose, check what actually filters incoming connections.

What Did We Learn?

Lessons Learned

- **Verify by reading logs and exit statuses, not by trusting schedules.** Every backup failure in this post was already recorded by the scheduler or in a log. None was announced. A job that is scheduled, a job that ran, and a job that covered what it should are three different claims.
- **A fix is not done until it has run on every machine it touches.** The lock fix was correct and broke joe. The address fix was correct on two machines and missed the third. The status check that found the outage covered the Macs and missed the Linux laptop.
- **The blast radius of a fix is not always one machine.** OAuth revocations, shared scripts, and broad symlink rules act on everything that uses the same mechanism. Know which kind of fix is in hand before applying it.
- **Rebuilding a machine from bare metal is the cheapest audit available,** because a fresh machine has no accumulated exceptions hiding real gaps.
- **Write it down.** Every finding went into the fleet's internal reference the day it was fixed, which is why the second week's failures were diagnosed in minutes rather than hours.

Limitations

- The audit checked the architecture this series describes. It was not a penetration test and does not claim to have found every gap.
- Several findings came from incidents observed directly, not from a systematic check, so there is no guarantee every instance of those problems has been found.
- The reinstall runbook has been exercised on one machine.
- Every failure here was found by a person looking. Until an automated check exists, the backups are only as verified as the last time someone read their status.

Opportunities for Improvement

1. A daily status check that reports, for every machine including the Linux ones, each backup job's loaded state, last exit status, and log freshness, and flags anything older than a day or not successful.
2. Backup scripts that wait or retry for network connectivity before failing.
3. A scheduled restore test, so that “backed up” means “restored at least once.”
4. Secret scanning on every commit to the dotfiles repository.
5. A second exercise of the reinstall runbook, to find the parts that were specific to the first machine by accident.

Wrapping Up

The audit the hub recommends was worth doing, though not in the shape expected. Every control the threat model named held. The failures lived around those controls: credentials with too little coverage or too much, OAuth grants whose fate is shared across machines, backups that succeeded while covering less than intended, and the slow drift of everything near a correct configuration, from a router-assigned address to a tool one machine never installed. None of these were configuration errors when they were configured. All of them were found by checking, and none by an alert.

Takeaways:

- Read backup logs and exit statuses on every machine; a schedule is not evidence.
- Before revoking any credential or grant, know everything that depends on it.
- Verify every fix on every machine it touches, including the ones that run a different operating system.
- Prefer addresses and configurations that cannot change underneath you, and know every file that names a machine.
- Record each finding in writing the day it is fixed.

See Also

Security series:

1. [Security Foundations for a Multi-Machine Research Fleet](#)

2. [Setting Up pass: a Unix Password Manager](#)
3. [Secrets Management for the Workflow Construct](#)
4. [Migrating Off Dropbox: Beyond Dotfiles](#)
5. [Research Backup Architecture: Borg, Cloud Copies, and Time Machine](#)
6. [A Home Server for a Five-Machine Research Fleet](#)
7. [Object Storage for Research Data: The Layer Git Cannot Hold](#)
8. [Running the Quarterly Security Audit for the First Time: What Actually Broke \(this post\)](#)

External references:

- [BorgBackup documentation](#)
- [rclone documentation](#)
- [GitHub CLI manual](#)
- [Tailscale documentation](#)
- [Samba vfs_fruit manual page](#)

Reproducibility

This post contains no executable analysis. The commands shown are the checks and the lock line used on the fleet during the audit. Environment at the time of writing: macOS 26.6 on the three Macs, Linux Mint 22.3 on the Linux laptop and Linux Mint on the home server, Borg 1.4.5, rclone 1.75, flock 0.4.0 (Homebrew). Exact command output, host names beyond the machine names, and addresses are omitted deliberately.

Let's Connect

Questions, suggestions, or corrections are welcome.

- **GitHub:** [rgt47](#)
- **Twitter/X:** [@rgt47](#)
- **LinkedIn:** [Ronald Glenn Thomas](#)
- **Email:** [Contact form](#)

I would like to hear from you if:

- You find an error, or a better approach to anything in this series.
- You run a small research fleet and have found where this design breaks.
- You want to compare notes on backups, secrets, or reproducible research infrastructure.